Graph (1)

Nov 22nd, 2018

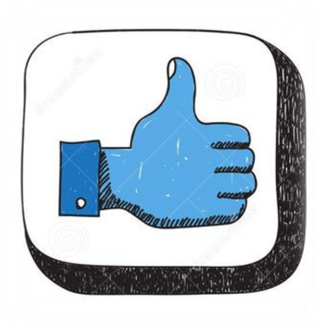
Algorithm Design and Analysis

YUN-NUNG (VIVIAN) CHEN HTTP://ADA.MIULAB.TW

國立臺灣大學 National Taiwan University

Slides credited from Hsueh-I Lu, Hsu-Chun Hsiao, & Michael Tsai

Midterm Feedback



- Mini-HW
- NTU COOL
- TA hours
- Course recordings
- Instant feedback

- Classroom (crowded, sleepy, etc.)
- Homework due time
- Pseudo code
- Difficulty of homework & exam
- TA recitation
- Seat announcement

Announcement

- Mini-HW 7 released
  - Due on 11/29 (Thur) 14:20
- Homework 3 released soon
  - Due on 12/13 (Thur) 14:20 (three weeks)

Frequently check the website for the updated information!

Mini-HW 7

Given a tree with N nodes, where each edge of the tree is weighted with W_i .

(1) Please design an algorithm (that runs in O(N) time) to accumulate the weights of all edges linking u and v. (For this question, a clear explanation is enough, no pseudo code is needed)

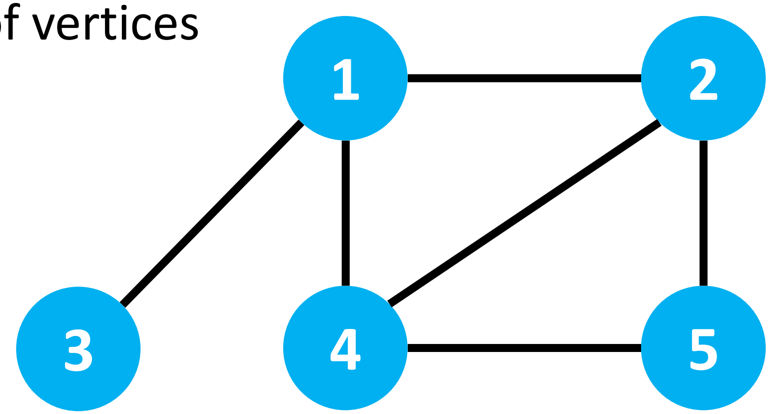
(2) Please simply justify the correctness of your algorithm.

Outline

- Graph Basics
- Graph Theory
- Graph Representations
- Graph Traversal
  - Breadth-First Search (BFS)
  - Depth-First Search (DFS)
- DFS Applications
  - Connected Components
  - Strongly Connected Components
  - Topological Sorting

Graph Basics

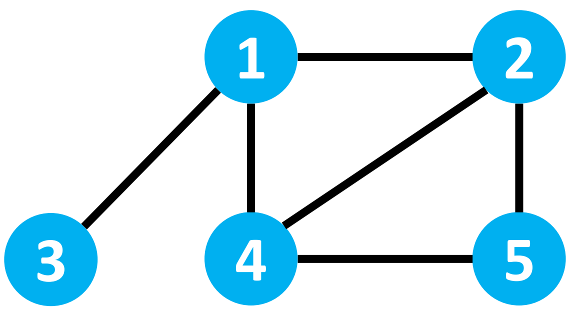
- A graph G is defined as $G = (V, E)$
  - V: a finite, nonempty set of vertices
  - E: a set of edges / pairs of vertices



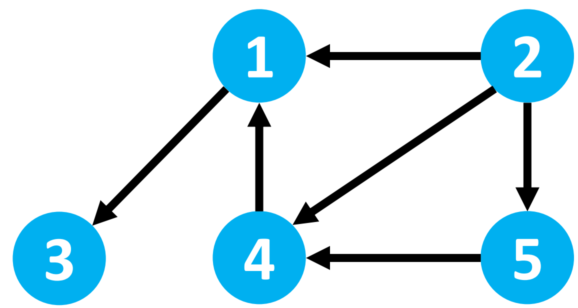
$$V = \{1, 2, 3, 4, 5\}$$
$$E = \{(1, 2), (1, 3), (1, 4), (2, 4), (2, 5), (4, 5)\}$$

Graph Basics

- Graph type
  - **Undirected**: edge $(u, v) = (v, u)$
  - **Directed**: edge (u, v) goes from vertex u to vertex v; $(u, v) \neq (v, u)$
  - **Weighted**: edges associate with weights



$$V = \{1, 2, 3, 4, 5\}$$
$$E = \{(1, 2), (1, 3), (1, 4), (2, 4), (2, 5), (4, 5)\}$$



$$V = \{1, 2, 3, 4, 5\}$$
$$E = \{(2, 1), (1, 3), (4, 1), (2, 4), (2, 5), (5, 4)\}$$

How many edges at most can a undirected (or directed) graph have?

Graph Basics

- **Adjacent** (相鄰)
  - If there is an edge (u, v), then u and v are adjacent.
- **Incident** (作用)
  - If there is an edge (u, v), the edge(u, v) is incident from u and is incident to v.
- **Subgraph** (子圖)
  - If a graph $G' = (V', E')$ is a subgraph of $G = (V, E)$, then $V' \subseteq V$ and $E' \subseteq E$

Graph Basics

- **Degree**
  - The degree of a vertex u is the number of edges incident on u
    - In-degree of u: #edges (x, u) in a directed graph
    - Out-degree of u: #edges (u, x) in a directed graph
    - Degree = in-degree + out-degree
    - **Isolated** vertex: degree = 0

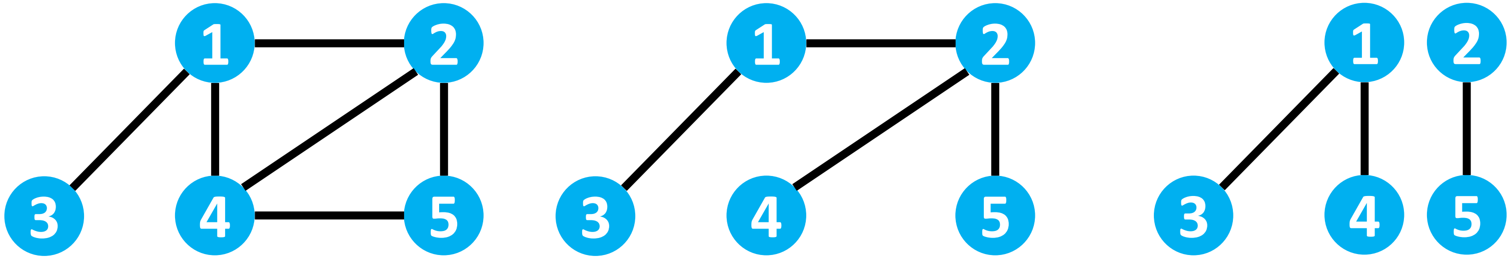
$$|E| = \frac{(\sum_i d_i)}{2}$$

Graph Basics

- **Path**
  - a sequence of edges that connect a sequence of vertices
  - If there is a path from u (source) to v (target), there is a sequence of edges $(u, i_1), (i_1, i_2), \ldots, (i_{k-1}, i_k), (i_k, v)$
  - **Reachable**: v is reachable from u if there exists a path from u to v
- **Simple Path**
  - All vertices except for u and v are all distinct
- **Cycle**
  - A simple path where u and v are the same
- **Subpath**
  - A subsequence of the path

Graph Basics

- **Connected**
  - Two vertices are connected if there is a path between them
  - A connected graph has a path from every vertex to every other
- **Tree**
  - a connected, acyclic, undirected graph
- **Forest**
  - an acyclic, undirected but possibly disconnected graph



Graph Basics

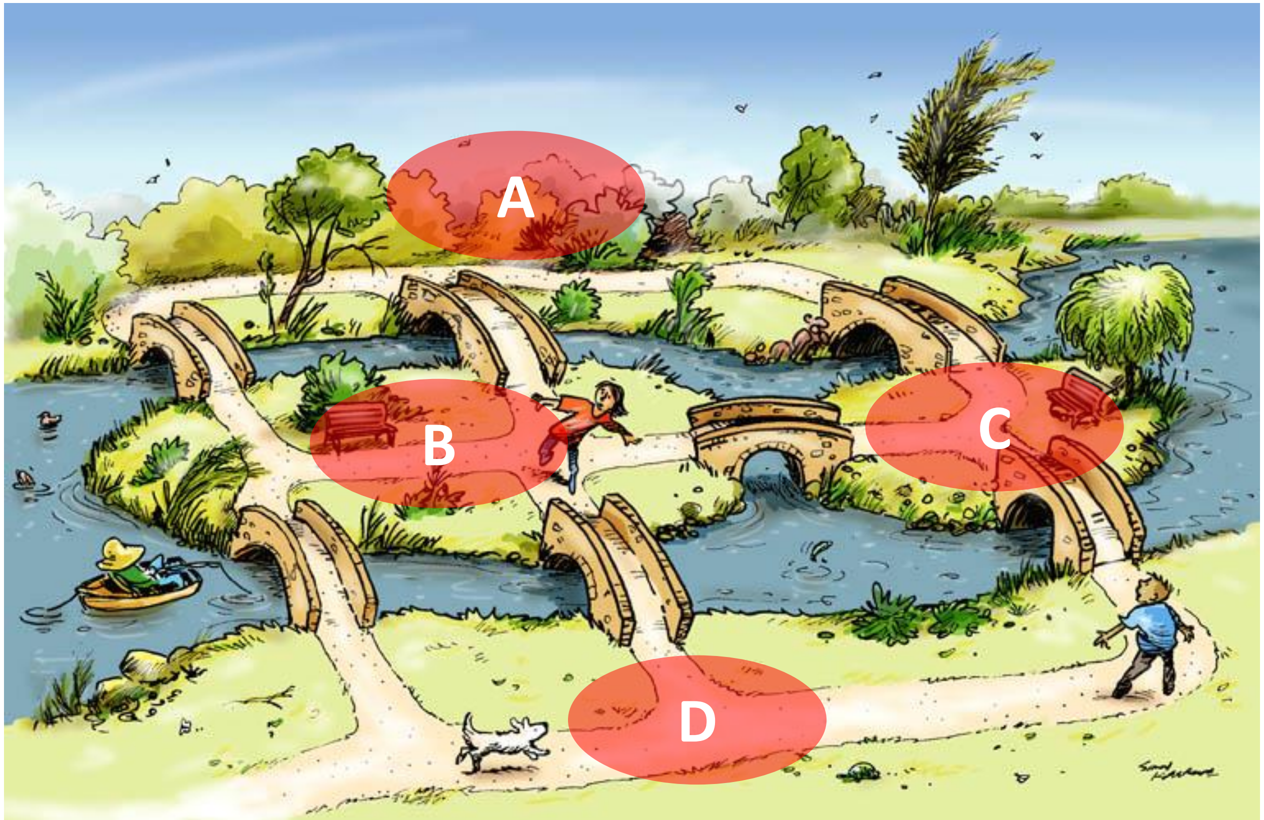
- Theorem. Let G be an undirected graph. The following statements are equivalent:
  - G is a tree
  - Any two vertices in G are connected by a unique simple path
  - G is connected, but if any edge is removed from E, the resulting graph is disconnected.
  - G is connected and $|E| = |V| - 1$
  - G is acyclic, and $|E| = |V| - 1$
  - G is acyclic, but if any edge is added to E, the resulting graph contains a cycle

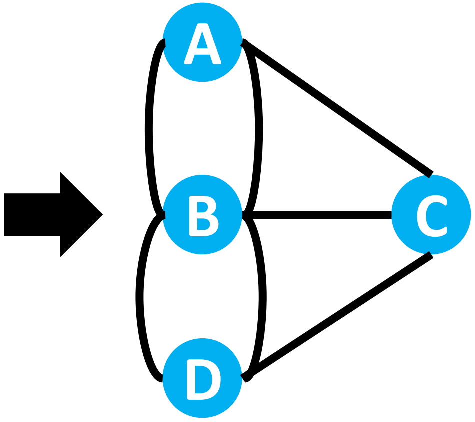
Proofs in Textbook Appendix B.5

13 Graph Theory

Seven Bridges of Königsberg (七橋問題)

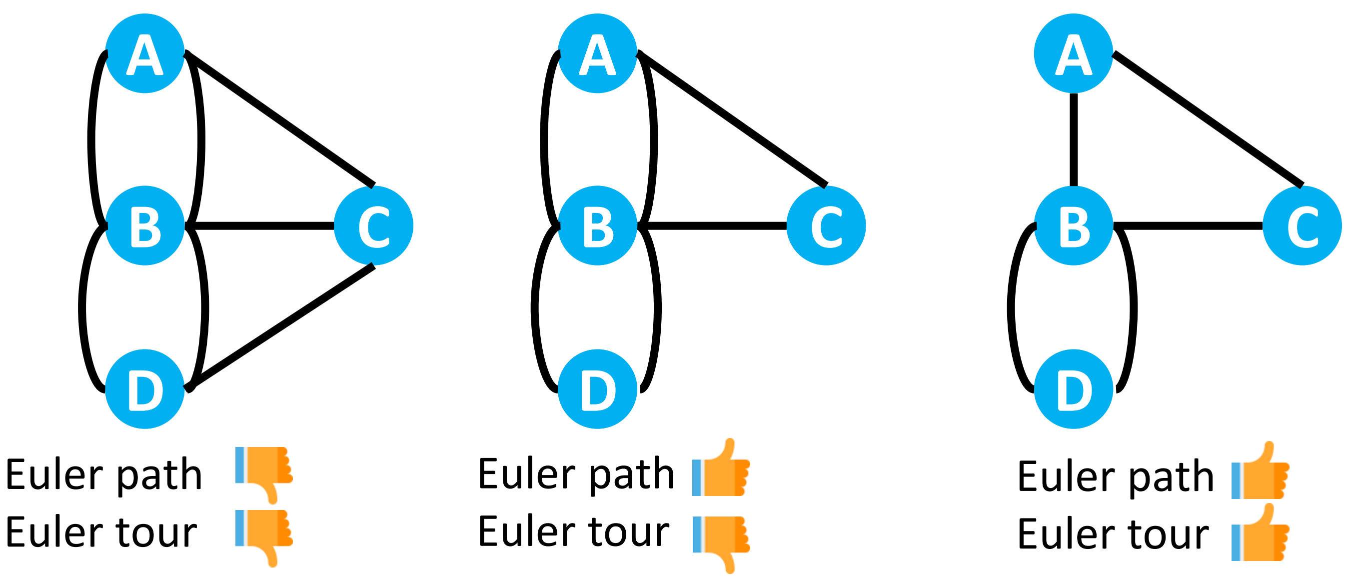
- How to traverse all bridges where each one can only be passed through once





Euler Path and Euler Tour (一筆畫問題)

- Euler path
  - Can you traverse each edge in a connected graph exactly once without lifting the pen from the paper?
- Euler tour
  - Can you finish where you started?



Euler Path and Euler Tour

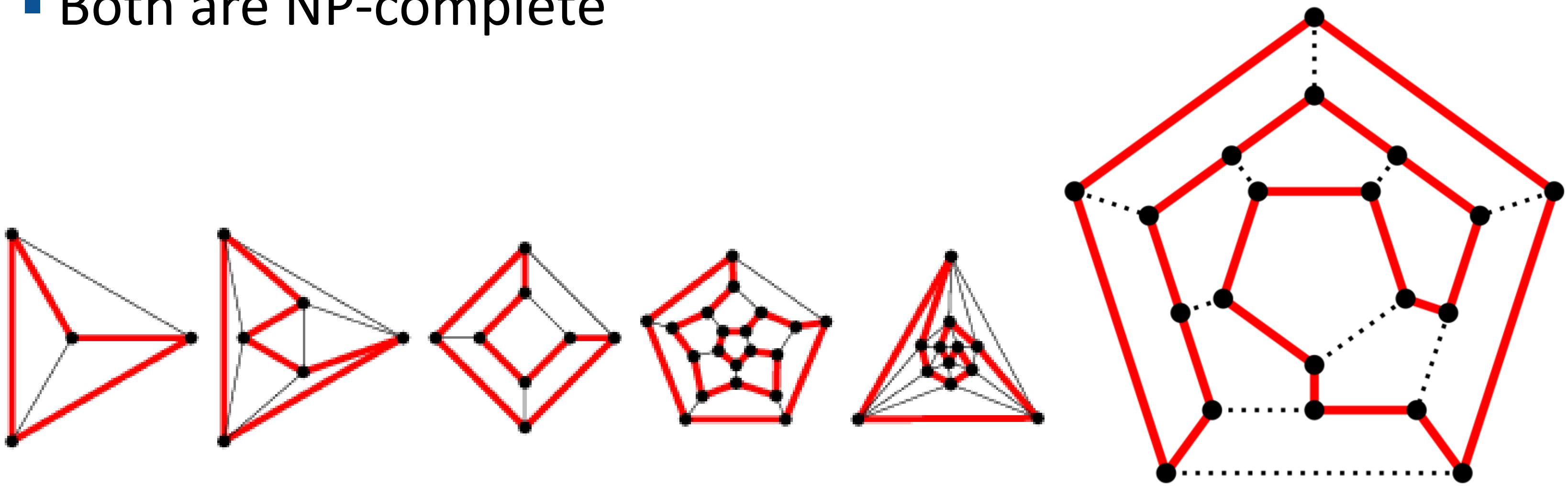
Is it possible to determine whether a graph has an *Euler path* or an *Euler tour*, without necessarily having to find one explicitly?

- Solved by Leonhard Euler in 1736
- G has an Euler path iff G has exactly 0 or 2 odd vertices
- G has an Euler tour iff all vertices must be even vertices

Even vertices = vertices with even degrees
Odd vertices = vertices with odd degrees

Hamiltonian Path

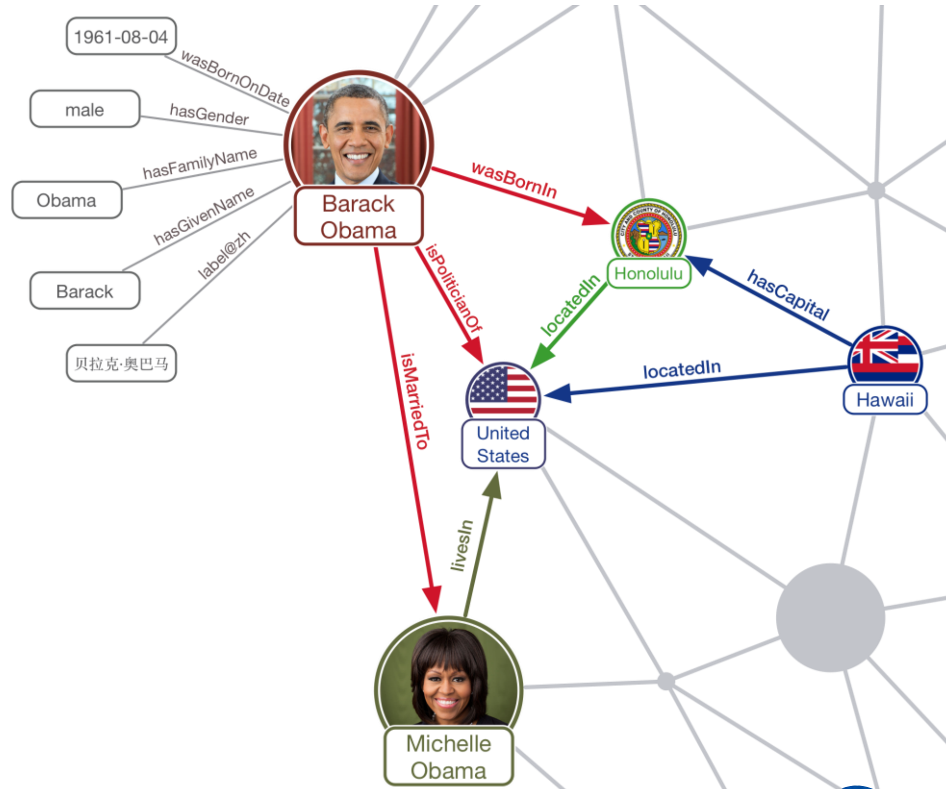
- Hamiltonian Path
  - A path that visits each vertex exactly once
- Hamiltonian Cycle
  - A Hamiltonian path where the start and destination are the same
- Both are NP-complete



Real-World Applications

- Modeling applications using graph theory
  - What do the vertices represent?
  - What do the edges represent?
  - Undirected or directed?

Social Network



Knowledge Graph

19

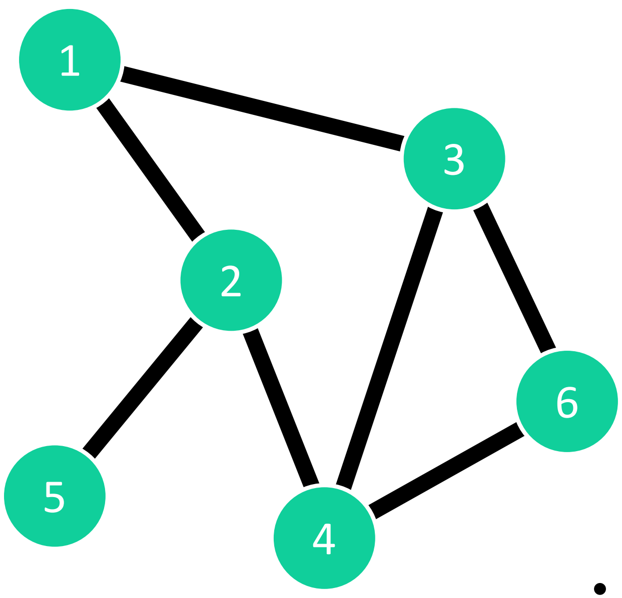
Graph Representations

Graph Representations

- How to represent a graph in computer programs?
- Two standard ways to represent a graph $G = (V, E)$
  - Adjacency matrix
  - Adjacency list

Adjacency Matrix

- Adjacency matrix = $V \times V$ matrix A with $A[u][v] = 1$ if (u, v) is an edge



	1	2	3	4	5	6
1		1	1			
2	1			1	1	
3	1			1		1
4		1	1			1
5		1				
6			1	1		

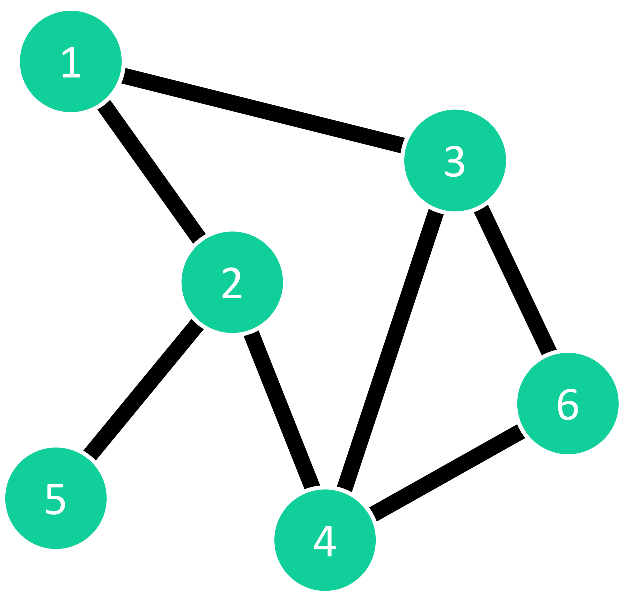
- For undirected graphs, A is symmetric; i.e., $A = A^T$
- If weighted, store weights instead of bits in A

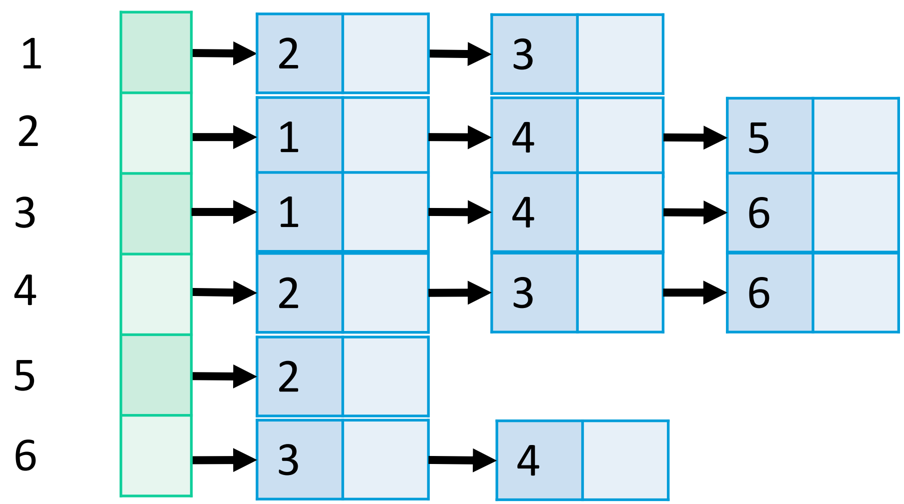
Complexity of Adjacency Matrix

- Space: $\Theta(n^2)$
- Time for querying an edge: $\Theta(1)$
- Time for inserting an edge: $\Theta(1)$
- Time for deleting an edge: $\Theta(1)$
- Time for listing all neighbors of a vertex: $\Theta(n)$
- Time for identifying all edges: $\Theta(n^2)$
- Time for finding in-degree and out-degree of a vertex?

Adjacency List

- Adjacency lists = vertex indexed array of lists
  - One list per vertex, where for $u \in V$, $A[u]$ consists of all vertices adjacent to u





If weighted, store weights also in adjacency lists

Complexity of Adjacency List

- Space: $\Theta(m+n)$
- Time for querying an edge: $\Theta(\deg)$ ➡ $\Theta(\log \deg)$
- Time for inserting an edge: $\Theta(1)$ ➡ $\Theta(\log \deg)$
- Time for deleting an edge: $\Theta(\deg)$ ➡ $\Theta(\log \deg)$
- Time for listing all neighbors of a vertex: $\Theta(\deg)$
- Time for identifying all edges: $\Theta(m+n)$
- Time for finding in-degree and out-degree of a vertex?

Representation Comparison

- Matrix representation is suitable for **dense** graphs
- List representation is suitable for **sparse** graphs
- Besides graph density, you may also choose a data structure based on the performance of other operations

	Space	Query an edge	Insert an edge	Delete an edge	List a vertex's neighbors	Identify all edges
Adjacency Matrix	$\Theta(n^2)$	$\Theta(1)$	$\Theta(1)$	$\Theta(1)$	$\Theta(n)$	$\Theta(n^2)$
Adjacency List	$\Theta(m+n)$	$\Theta(\deg)$ $\Theta(\log \deg)$	$\Theta(1)$ $\Theta(\log \deg)$	$\Theta(\deg)$ $\Theta(\log \deg)$	$\Theta(\deg)$	$\Theta(m+n)$

26 Graph Traversal

Textbook Chapter 22 – Elementary Graph Algorithms

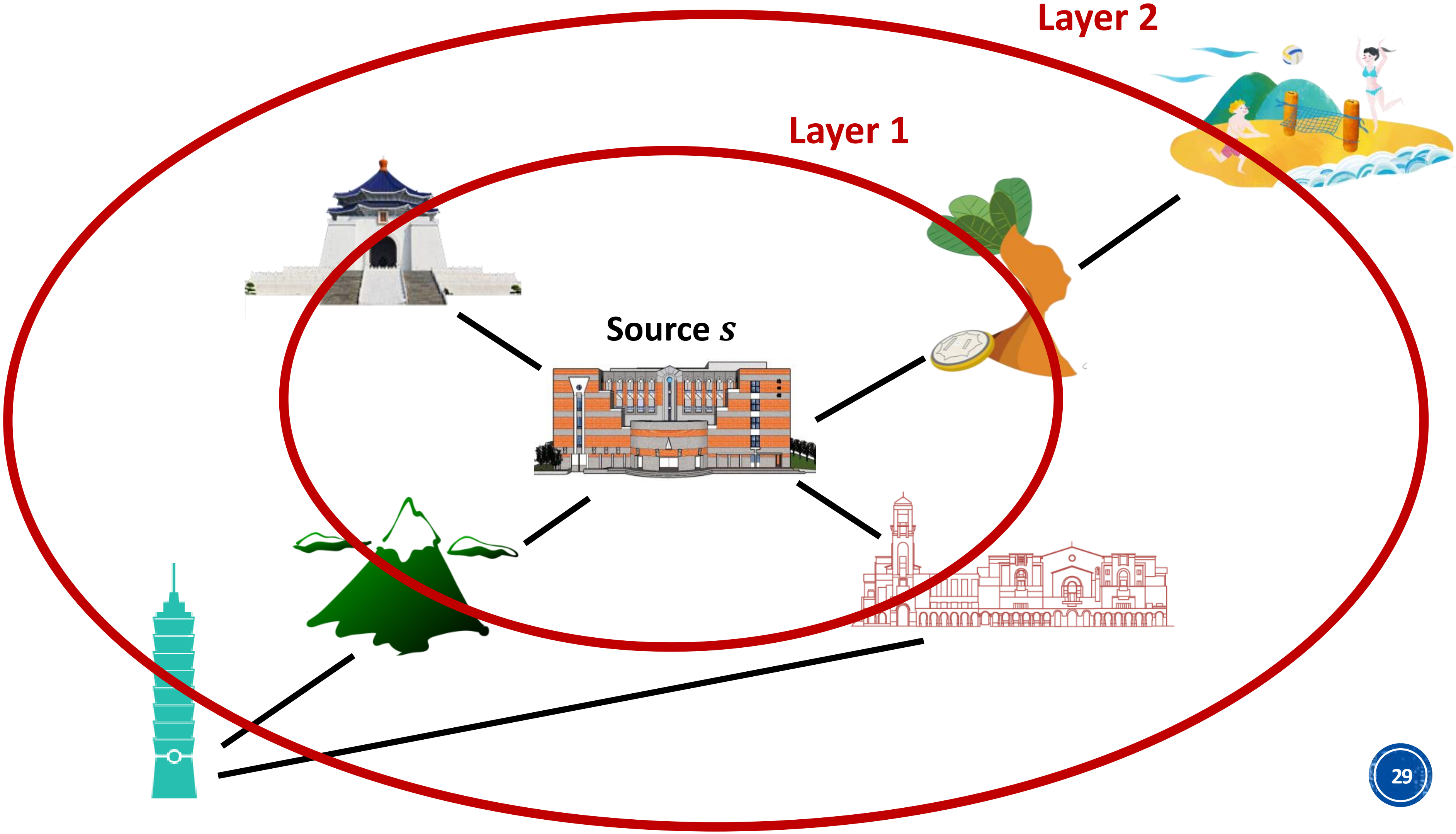
Graph Traversal

- From a source vertex, systematically follow the edges of a graph to visit all reachable vertices of the graph
- Useful to discover the structure of a graph
- Standard graph-searching algorithms
  - Breadth-First Search (BFS, 廣度優先搜尋)
  - Depth-First Search (DFS, 深度優先搜尋)

28 Breadth-First Search

Textbook Chapter 22.2 – Breadth-first search

Breadth-First Search (BFS)



Breadth-First Search (BFS)

- Input: directed/undirected graph $G = (V, E)$ and source s
- Output: a **breadth-first tree** with root s (T_{BFS}) that contains all reachable vertices
  - $v.d$: distance from s to v, for all $v \in V$
    - Distance is the length of a shortest path in G
    - $v.d = \infty$ if v is not reachable from s
    - $v.d$ is also the depth of v in T_{BFS}
  - $v.\pi = u$ if (u, v) is the last edge on shortest path to v
    - u is v's predecessor in T_{BFS}

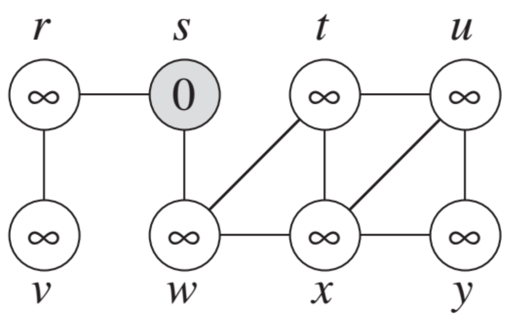
Breadth-First Tree

- Initially T_{BFS} contains only s
- As v is discovered from u, v and (u, v) are added to T_{BFS}
  - T_{BFS} is not explicitly stored; can be reconstructed from $v.\pi$
- Implemented via a FIFO queue
- Color the vertices to keep track of progress:
  - GRAY: discovered (first time encountered)
  - BLACK: finished (all adjacent vertices discovered)
  - WHITE: undiscovered

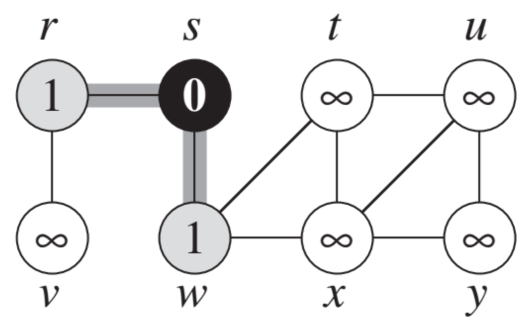
```
BFS(G, s)
  for each vertex u in G.V-{s}          O(n)
    u.color = WHITE
    u.d = ∞
    u.pi = NIL
  s.color = GRAY
  s.d = 0
  s.pi = NIL
  Q = {}
  ENQUEUE(Q, s)
  while Q! = {}
    u = DEQUEUE(Q)
    for each v in G.Adj[u]              O(deg(u))
      if v.color == WHITE
        v.color = GRAY
        v.d = u.d + 1
        v.pi = u
        ENQUEUE(Q,v)
    u.color = BLACK
```

$$\Rightarrow O\left(n + \sum_u (\deg(u) + 1)\right) = O(n + m)$$

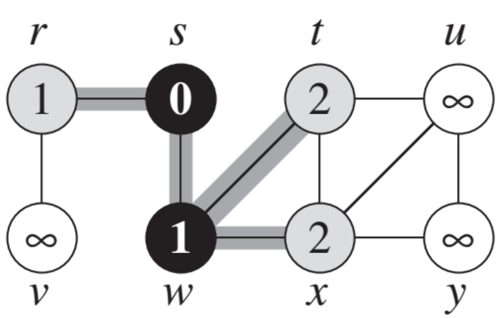
BFS Illustration



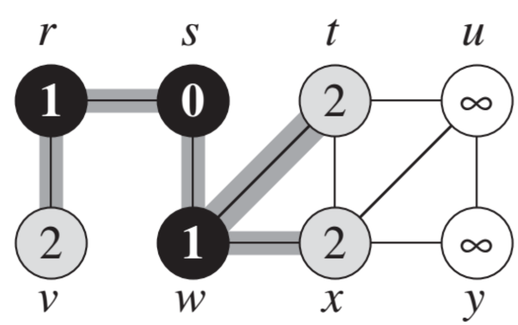
s
0



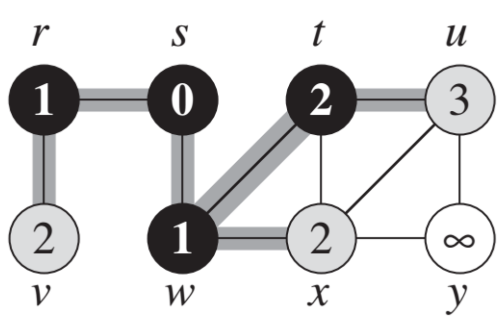
w	r
1	1



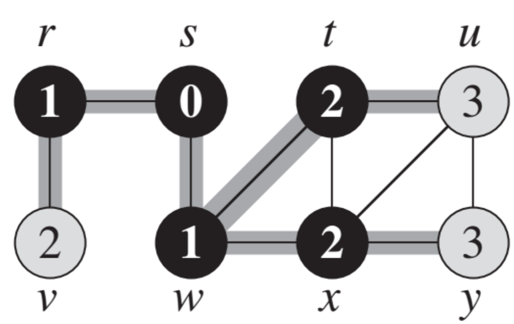
r	t	x
1	2	2



t	x	v
2	2	2

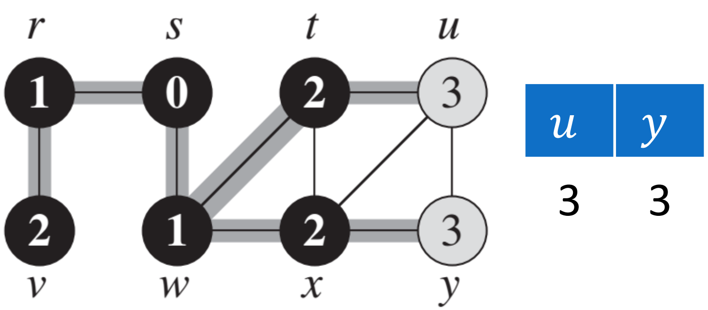


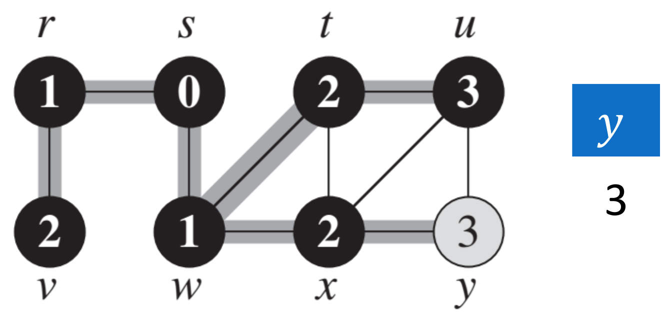
x	v	u
2	2	3

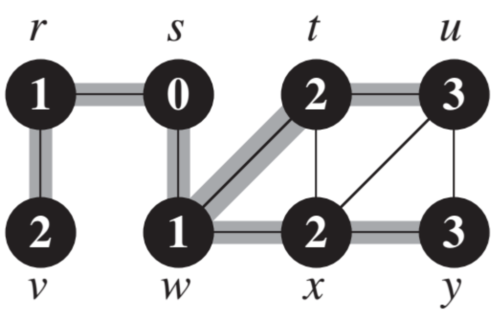


v	u	y
2	3	3

BFS Illustration







Shortest-Path Distance from BFS

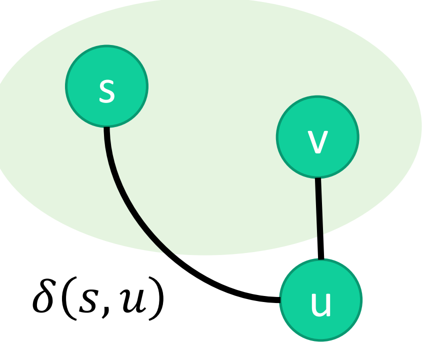
- Definition of $\delta(s, v)$: the **shortest-path distance** from s to v = the minimum number of edges in any path from s to v
  - If there is no path from s to v, then $\delta(s, v) = \infty$
- The BFS algorithm finds the shortest-path distance to each reachable vertex in a graph G from a given source vertex $s \in V$.

Shortest-Path Distance from BFS

Lemma 22.1
Let $G = (V, E)$ be a directed or undirected graph, and let $s \in V$ be an arbitrary vertex. Then, for any edge $(u, v) \in E$, $\delta(s, v) \leq \delta(s, u) + 1$.

s-v的最短路徑一定會小於等於s-u的最短路徑距離+1

- Proof
  - Case 1: u is reachable from s
    - s- u- v is a path from s to v with length $\delta(s, u) + 1$
    - Hence, $\delta(s, v) \leq \delta(s, u) + 1$
  - Case 2: u is unreachable from s
    - Then v must be unreachable too.
    - Hence, the inequality still holds.



Shortest-Path Distance from BFS

<u>Lemma 22.2</u>
Let $G = (V, E)$ be a directed or undirected graph, and suppose BFS is run on G from a given source vertex $s \in V$. Then upon termination, for each vertex $v \in V$, the value $v.d$ computed by BFS satisfies $v.d \geq \delta(s, v)$.

- Proof by induction

BFS算出的d值必定大於等於真正距離

Inductive hypothesis: $v.d \geq \delta(s, v)$ after n ENQUEUE ops

  - Holds when $n = 1$: s is in the queue and $v.d = \infty$ for all $v \in V\{s\}$
  - After $n + 1$ ENQUEUE ops, consider a white vertex v that is discovered during the search from a vertex u

$$\begin{aligned} v.d = u.d + 1 &\geq \delta(s, u) + 1 && \text{(by induction hypothesis)} \\ &\geq \delta(s, v) && \text{(by Lemma 22.1)} \end{aligned}$$

  - Vertex v is never enqueued again, so $v.d$ never changes again

Shortest-Path Distance from BFS

Lemma 22.3

Suppose that during the execution of BFS on a graph $G = (V, E)$, the queue Q contains the vertices $\langle v_1, v_2, \dots, v_r \rangle$, where v_1 is the head of Q and v_r is the tail. Then, $v_r.d \le v_1.d + 1$ and $v_i.d \le v_{i+1}.d$ for $1 \le i < r$.

- Q中最後一個點的d值 ≤ Q中第一個點的d值+1
- Q中第i個點的d值 ≤ Q中第i+1點的d值

- Proof by induction

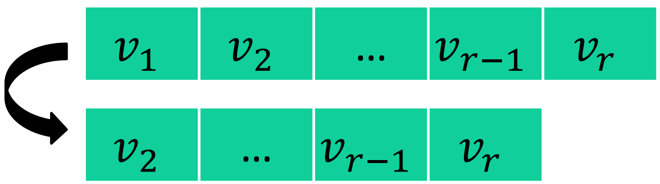
Inductive hypothesis: $v_r.d \le v_1.d + 1$ and $v_i.d \le v_{i+1}.d$ after n queue ops

  - Holds when $Q = \langle s \rangle$.
  - Consider two operations for inductive step:
    - Dequeue op: when $Q = \langle v_1, v_2, \dots, v_r \rangle$ and dequeue v_1
    - Enqueue op: when $Q = \langle v_1, v_2, \dots, v_r \rangle$ and enqueue v_{r+1}

Shortest-Path Distance from BFS

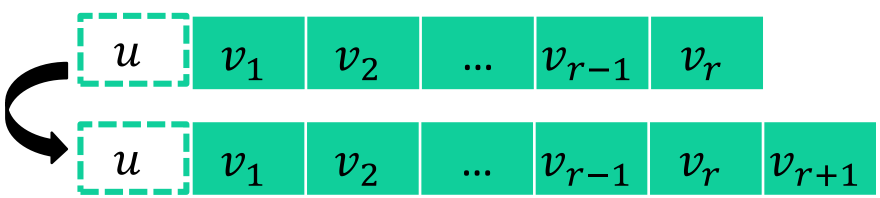
Inductive hypothesis: H1 $v_r.d \le v_1.d + 1$ (Q中最後一個點的d值 ≤ Q中第一個點的d值+1)
H2 $v_i.d \le v_{i+1}.d, i = 1, \cdots, r-1$ (Q中第i個點的d值 ≤ Q中第i+1點的d值)

- Dequeue op



$v_r.d \le v_1.d + 1$ (induction hypothesis H1)
$\le v_2.d + 1$ (induction hypothesis H2) → H1 holds
$v_i.d \le v_{i+1}.d, i = 2, \cdots, r-1$ → H2 holds

- Enqueue op



Let u be v_{r+1}'s predecessor, $v_{r+1}.d = u.d + 1$
Since u has been removed from Q, the new head v_1 satisfies $u.d \le v_1.d$ (induction hypothesis H2)
$v_{r+1}.d \le u.d + 1 \le v_1.d + 1$ → H1 holds
$v_r.d \le u.d + 1$ (induction hypothesis H1)
$v_r.d \le u.d + 1 = v_{r+1}.d$
$v_i.d = v_{i+1}.d, i = 1, \cdots, r$ → H2 holds

Shortest-Path Distance from BFS

<u>Corollary 22.4</u>
Suppose that vertices v_i and v_j are enqueued during the execution of BFS, and that v_i is enqueued before v_j. Then $v_i.d \leq v_j.d$ at the time that v_j is enqueued.

- Proof

  若v_i比v_j早加入queue → $v_i.d \leq v_j.d$

  - Lemma 22.3 proves that $v_i.d \leq v_{i+1}.d$ for $1 \leq i < r$
  - Each vertex receives a finite d value at most once during the course of BFS
  - Hence, this is proved.

Shortest-Path Distance from BFS

Theorem 22.5 – BFS Correctness

Let $G = (V, E)$ be a directed or undirected graph, and and suppose that BFS is run on G from a given source vertex $s \in V$.

1) BFS discovers every vertex $v \in V$ that is reachable from the source s
2) Upon termination, $v.d = \delta(s, v)$ for all $v \in V$
3) For any vertex $v \neq s$ that is reachable from s, one of the shortest paths from s to v is a shortest path from s to $v.\pi$ followed by the edge $(v.\pi, v)$

- Proof of (1)
  - All vertices v reachable from s must be discovered; otherwise they would have $v.d = \infty > \delta(s, v)$. (contradicting with Lemma 22.2)

Shortest-Path Distance from BFS

(2) $v.d = \delta(s, v) \; \forall \; v \in V$

- Proof of (2) by contradiction
  - Assume some vertices receive d values not equal to its shortest-path distance
  - Let v be the vertex with minimum $\delta(s, v)$ that receives such an incorrect d value; clearly $v \neq s$
  - By Lemma 22.2, $v.d \geq \delta(s, v)$, thus $v.d > \delta(s, v)$ (v must be reachable)
  - Let u be the vertex immediately preceding v on a shortest path from s to v, so $\delta(s, v) = \delta(s, u) + 1$
  - Because $\delta(s, u) < \delta(s, v)$ and v is the minimum $\delta(s, v)$, we have $u.d = \delta(s, u)$
  - $v.d > \delta(s, v) = \delta(s, u) + 1 = u.d + 1$

Shortest-Path Distance from BFS

(2) $v.d = \delta(s, v) \; \forall \; v \in V$

- Proof of (2) by contradiction (cont.)
  - $v.d > \delta(s,v) = \delta(s,u) + 1 = u.d + 1$
  - When dequeuing u from Q, vertex v is either WHITE, GRAY, or BLACK
    - WHITE: $v.d = u.d + 1$, contradiction
    - BLACK: it was already removed from the queue
      - By Corollary 22.4, we have $v.d \leq u.d$, contradiction
    - GRAY: it was painted GRAY upon dequeuing some vertex w
      - Thus $v.d = w.d + 1$ (by construction)
      - w was removed from Q earlier than u, so $w.d \leq u.d$ (by Corollary 22.4)
      - $v.d = w.d + 1 \leq u.d + 1$, contradiction
  - Thus, (2) is proved.

Shortest-Path Distance from BFS

(3) For any vertex $v \neq s$ that is reachable from s, one of the shortest paths from s to v is a shortest path from s to $v.\pi$ followed by the edge $(v.\pi, v)$

- Proof of (3)
  - If $v.\pi = u$, then $v.d = u.d + 1$. Thus, we can obtain a shortest path from s to v by taking a shortest path from s to $v.\pi$ and then traversing the edge $(v.\pi, v)$.

BFS Forest

- `BFS(G, s)` forms a BFS tree with all reachable v from s
- We can extend the algorithm to find a BFS forest that contains every vertex in G

```
//explore full graph and builds up
a collection of BFS trees
BFS(G)
  for u in G.V
    u.color = WHITE
    u.d = ∞
    u.π = NIL
  for s in G.V
    if(s.color == WHITE)
      // build a BFS tree
      BFS-Visit(G, s)
```

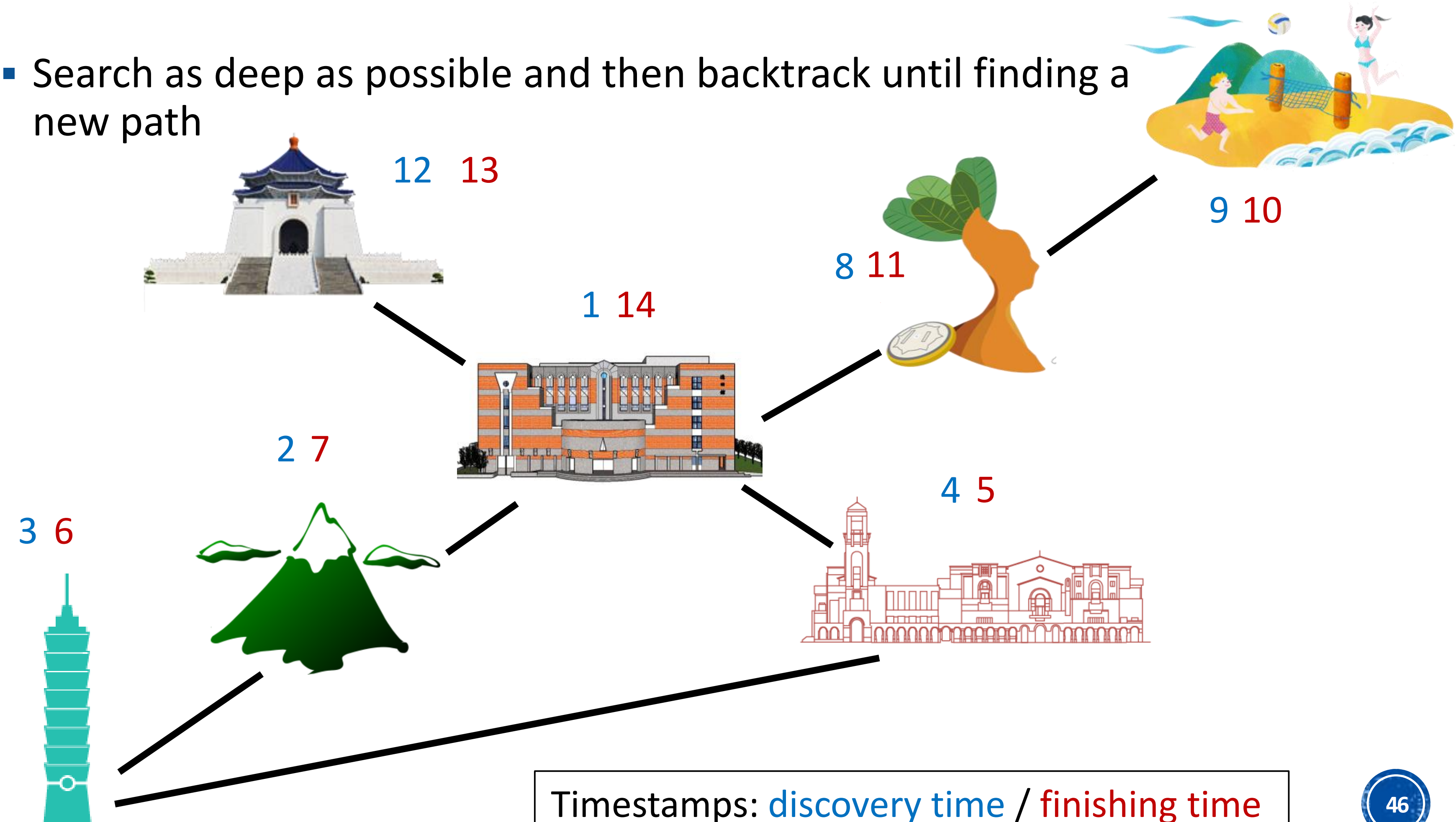
```
BFS-Visit(G, s)
  s.color = GRAY
  s.d = 0
  s.π = NIL
  Q = empty
  ENQUEUE(Q, s)
  while Q ≠ empty
    u = DEQUEUE(Q)
    for v in G.adj[u]
      if v.color == WHITE
        v.color = GRAY
        v.d = u.d + 1
        v.π = u
        ENQUEUE(Q, v)
  u.color = BLACK
```

45 Depth-First Search

Textbook Chapter 22.3 – Depth-first search

Depth-First Search (DFS)

- Search as deep as possible and then backtrack until finding a new path



Timestamps: discovery time / finishing time

DFS Algorithm

```
// Explore full graph and builds up
a collection of DFS trees
DFS(G)
  for each vertex u in G.V
    u.color = WHITE
    u.pi = NIL
  time = 0 // global timestamp
  for each vertex u in G.V
    if u.color == WHITE
      DFS-VISIT(G, u)
```

$O(n)$

```
DFS-Visit(G, u)
  time = time + 1
  u.d = time  // discover time
  u.color = GRAY
  for each v in G.Adj[u]
    if v.color == WHITE
      v.pi = u
      DFS-VISIT(G, v)
  u.color = BLACK
  time = time + 1
  u.f = time // finish time
```

$O(\deg(u) + 1)$

- Implemented via recursion (stack)
- Color the vertices to keep track of progress:
  - GRAY: discovered (first time encountered)
  - BLACK: finished (all adjacent vertices discovered)
  - WHITE: undiscovered

$$\Rightarrow O\left(n + \sum_u (\deg(u) + 1)\right)$$

$$= O(n + m)$$

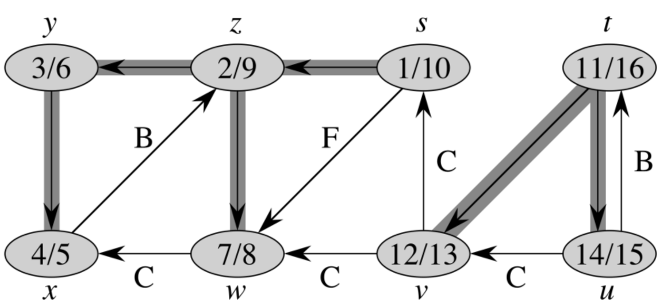
DFS Properties

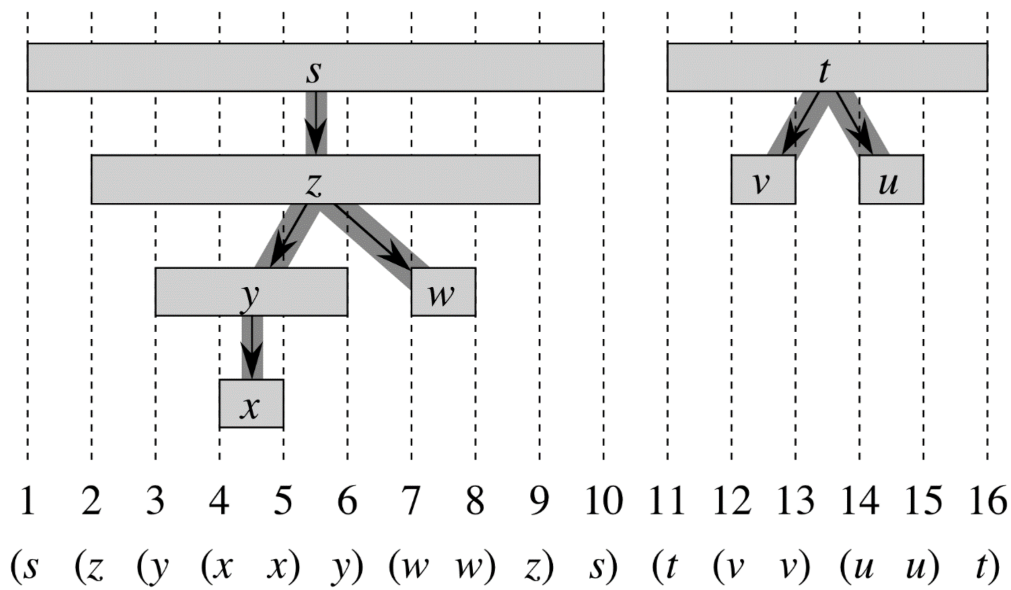
- Parenthesis Theorem
  - Parenthesis structure: represent the discovery of vertex u with a left parenthesis "(u" and represent its finishing by a right parenthesis "u)". In DFS, the parentheses are properly nested.
- White Path Theorem
  - In a DFS forest of a directed or undirected graph $G = (V, E)$,
    - vertex v is a descendant of vertex u in the forest ⇔ at the time $u.d$ that the search discovers u, there is a path from u to v in G consisting entirely of WHITE vertices
- Classification of Edges in G
  - Tree Edge
  - Back Edge
  - Forward Edge
  - Cross Edge

DFS Properties

- <u>Parenthesis Theorem</u>
  - Parenthesis structure: represent the discovery of vertex u with a left parenthesis "(u" and represent its finishing by a right parenthesis "u)". In DFS, the parentheses are properly nested.

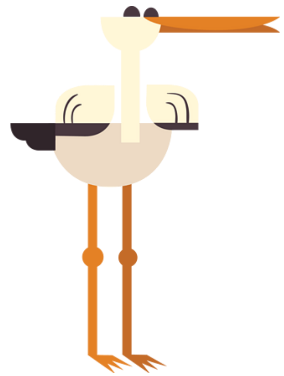
Properly nested: (x (y y) x)
Not properly nested: (x (y x) y)





Proof in textbook p. 608

DFS Properties



- <u>White Path Theorem</u>
  - In a DFS forest of a directed or undirected graph $G = (V, E)$,
    - vertex v is a descendant of vertex u in the forest ⇔ at the time $u.d$ that the search discovers u, there is a path from u to v in G consisting entirely of WHITE vertices
- Proof.
  - →
    - Since v is a descendant of u, $u.d < v.d$
    - Hence, v is WHITE at time $u.d$
    - In fact, since v can be any descendant of u, any vertex on the path from u to v are WHITE at time $u.d$
  - ← (textbook p. 608)

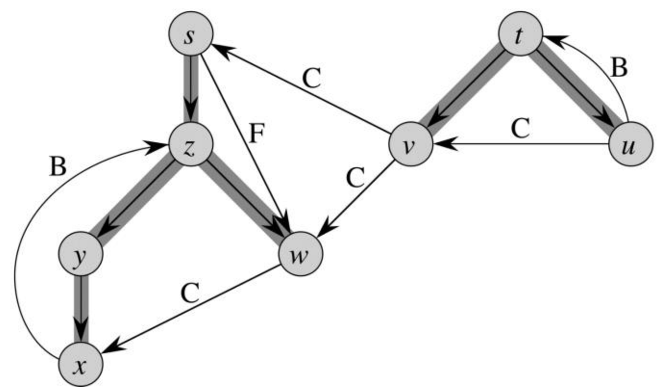
DFS Properties

- Classification of Edges in G
  - Tree Edge (GRAY to WHITE)
    - Edges in the DFS forest
    - Found when encountering a new vertex v by exploring (u, v)
  - Back Edge (GRAY to GRAY)
    - (u, v), from descendant u to ancestor v in a DFS tree
  - Forward Edge (GRAY to BLACK)
    - (u, v), from ancestor u to descendant v. Not a tree edge.
  - Cross Edge (GRAY to BLACK)
    - Any other edge between trees or subtrees. Can go between vertices in same DFS tree or in different DFS trees

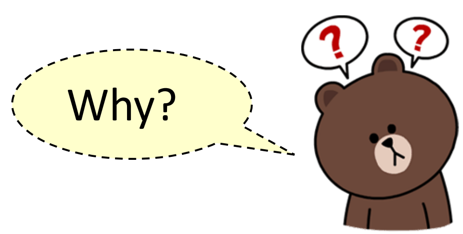
In an undirected graph, back edge = forward edge.
To avoid ambiguity, classify edge as the first type in the list that applies.

DFS Properties

- Edge classification by the color of v when visiting (u, v)
  - WHITE: tree edge
  - GRAY: back edge
  - BLACK: forward edge or cross edge
    - $u.d < v.d$ → forward edge
    - $u.d > v.d$ → cross edge



<u>Theorem 22.10</u>
In DFS of an undirected graph, there are only tree edges and back edges without forward and cross edge.



DFS Applications

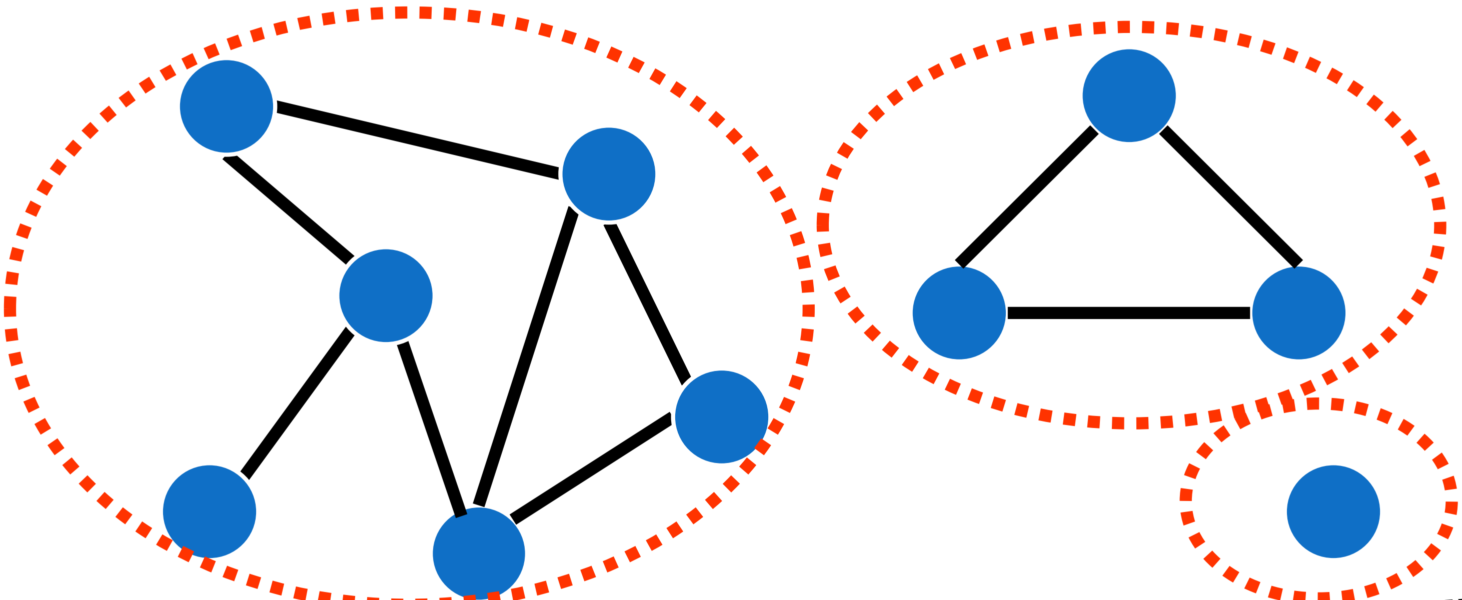
- Connected Components
- Strongly Connected Components
- Topological Sort

54

Connected Components

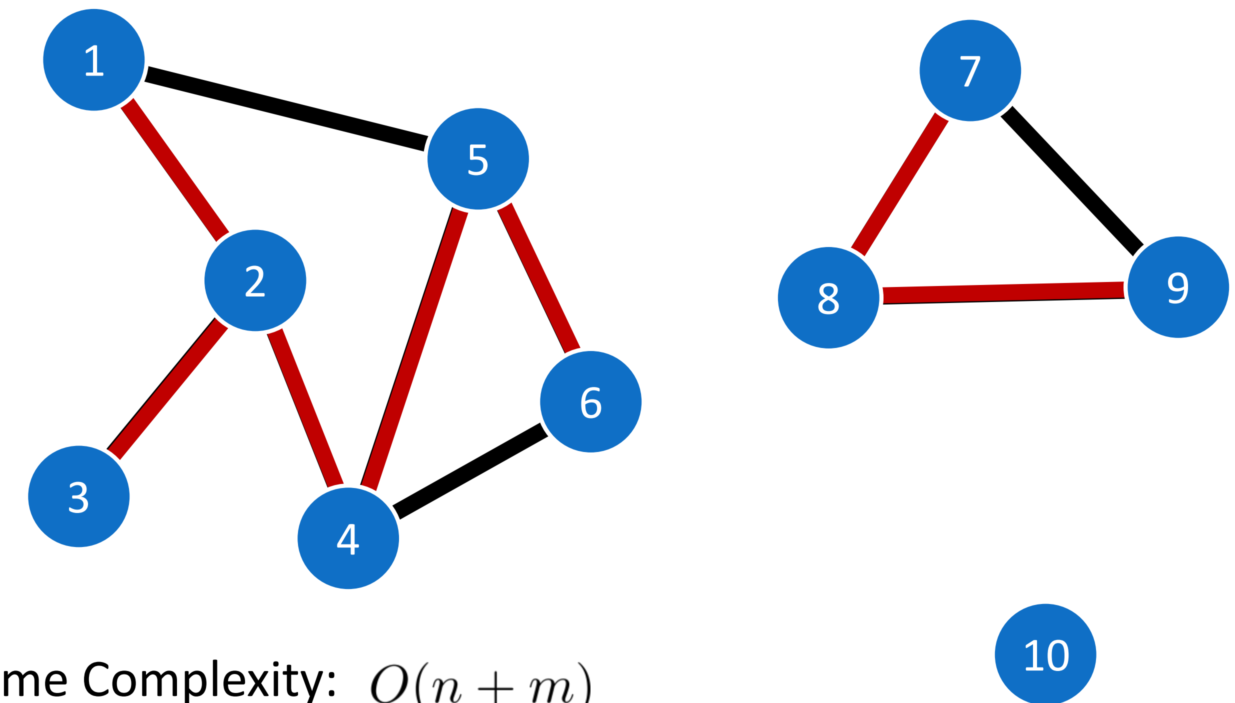
Connected Components Problem

- Input: a graph $G = (V, E)$
- Output: a connected component of G
  - a **maximal** subset U of V s.t. any two nodes in U are connected in G



Why must the connected components of a graph be disjoint?

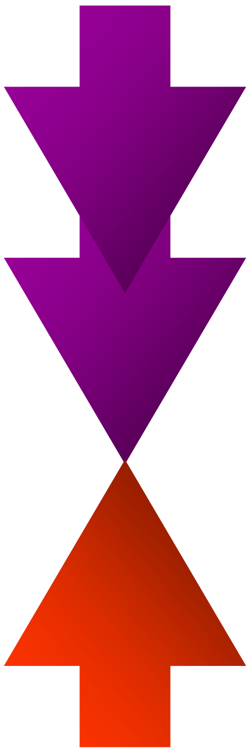
Connected Components



Time Complexity: $O(n+m)$

BFS and DSF both find the connected components with the same complexity

Problem Complexity



Upper bound $= O(m+n)$

Lower bound $= \Omega(m+n)$

To Be Continued…

Question?

Important announcement will be sent to @ntu.edu.tw mailbox
& post to the course website

Course Website: http://ada.miulab.tw

Email: ada-ta@csie.ntu.edu.tw